# Gas mentorship

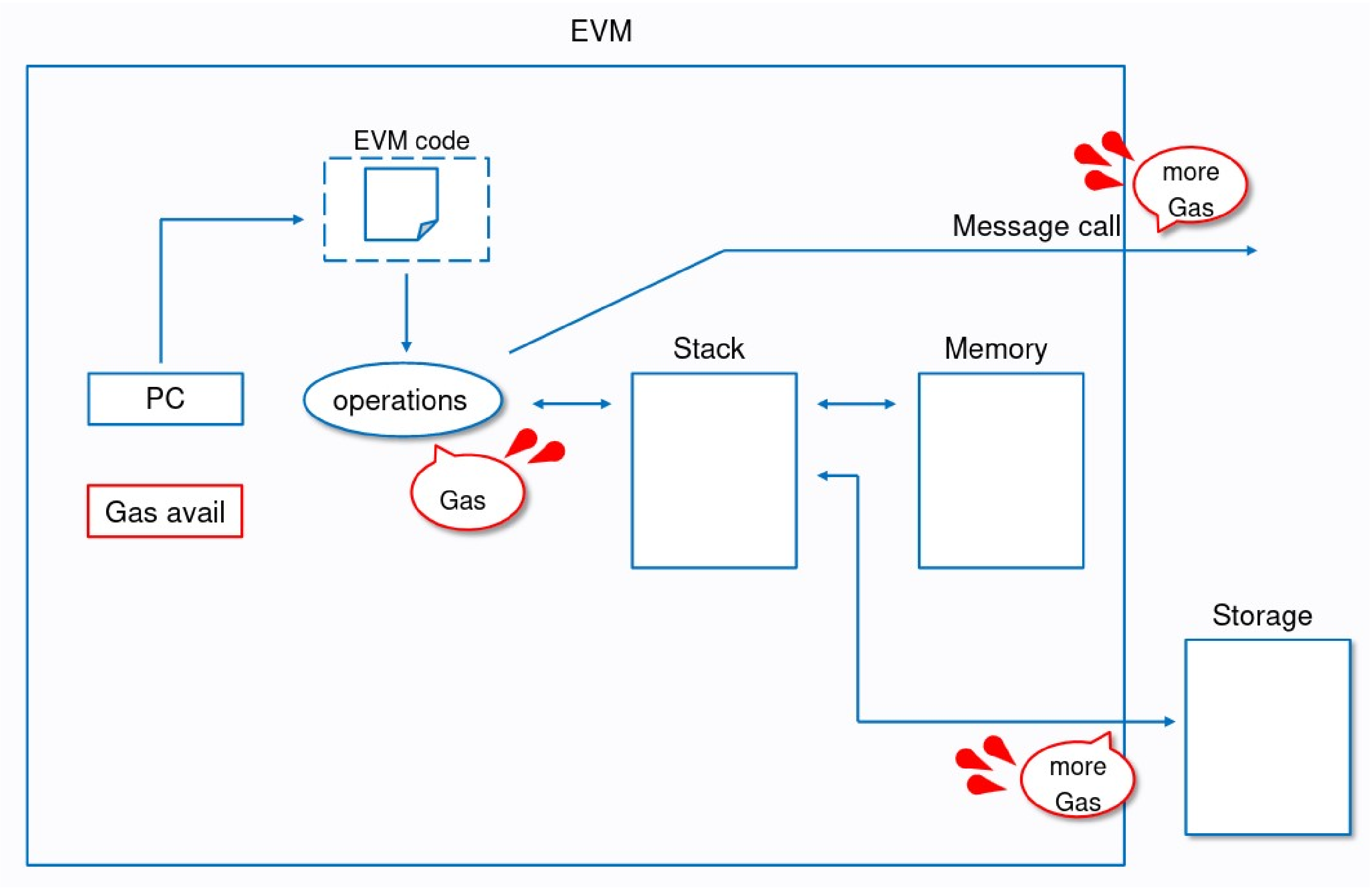
Comprehensive guide about gas optimization techniques

# Content

- Introduction
  - General overview of gas in ETH
- EVM Basics
  - Background knowledge about the EVM
- Gas optimizations
  - In depth review of gas optimizations

# Introduction

- What is gas?
  - Gas is the unit of computation in Ethereum
  - Each opcode has a fixed gas cost
  - Additionally, some opcodes have an dynamic gas cost on top of the fixed gas cost, e.g.
    - SLOAD/SSTORE
    - MLOAD/MSTORE


EVM
EVM code
more Gas
Message call
Stack
Memory
PC
operations
Gas
Gas avail
Storage
more Gas

# Introduction

- Each transaction has an intrinsic cost of 21k gas
- The more complex the transaction, the more expensive it is:
  - ETH Transfer: 21k gas
  - ERC20 Transfer: ~65k gas
  - ERC721 Transfer: ~84k gas
  - Uniswap V3 Swap: ~184k gas
  - Tornado Cash Deposit: ~1M gas

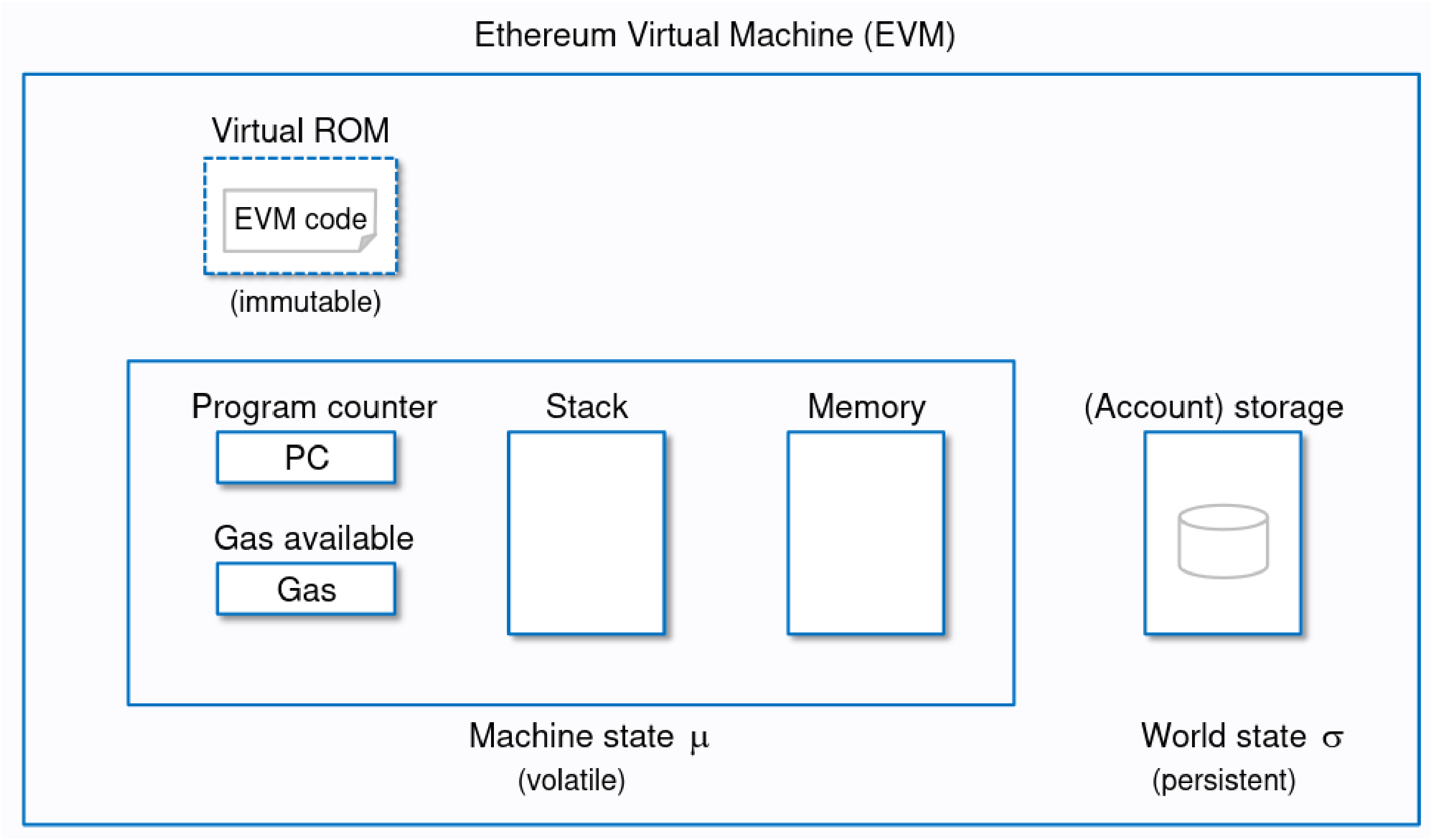
# Introduction

- How are gas costs calculated?
  - Since EIP-1559 the gas price consists of a
    - Base fee
    - Priority fee
  - The fee paid in USD is calculated as follows:

    (gas used x gas price (in gwei) x ether price (in USD)) / 1 billion

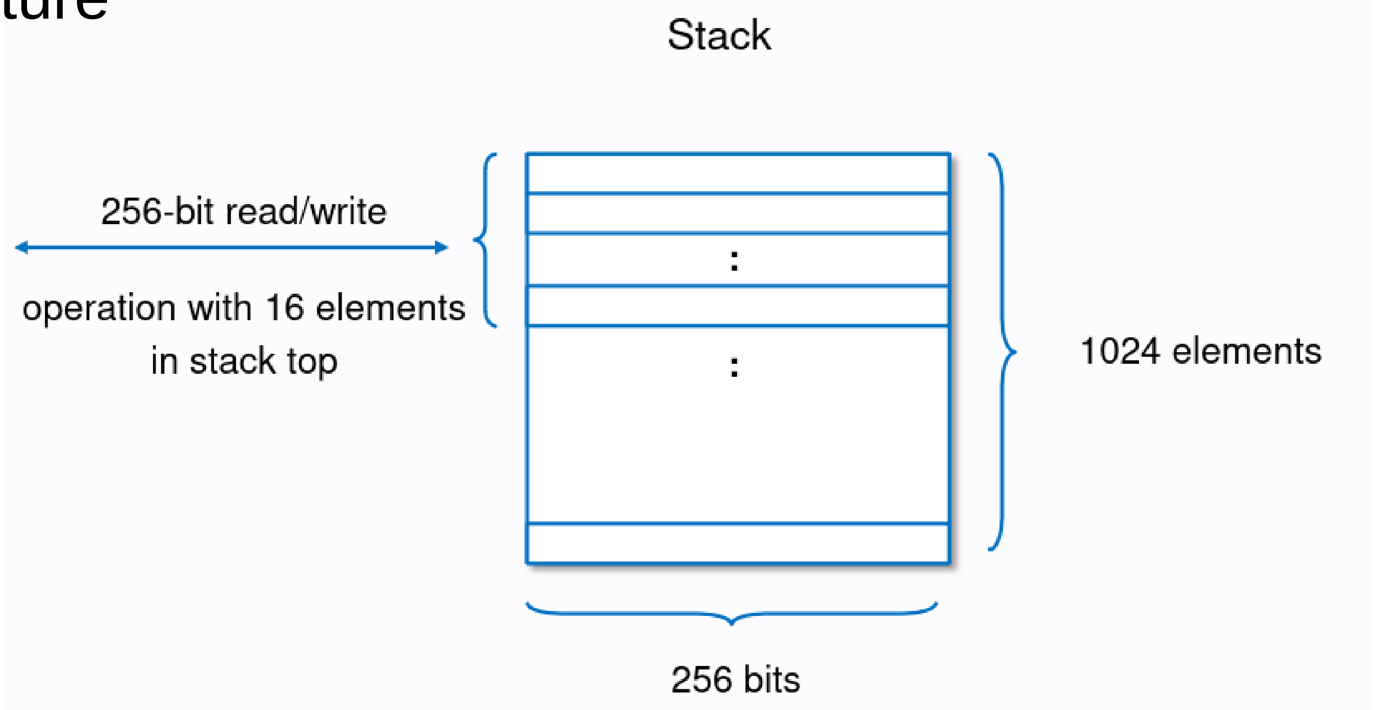
# EVM Basics

- Stack based computer
- Deterministic, always the same output for the same input
- Creates an execution context when executing a smart contract
  - Contains several data regions
  - Program counter (PC)
  - Gas available
  - ...


Ethereum Virtual Machine (EVM)
Virtual ROM
EVM code
(immutable)
Program counter
PC
Gas available
Gas
Stack
Memory
(Account) storage
Machine state μ
(volatile)
World state σ
(persistent)

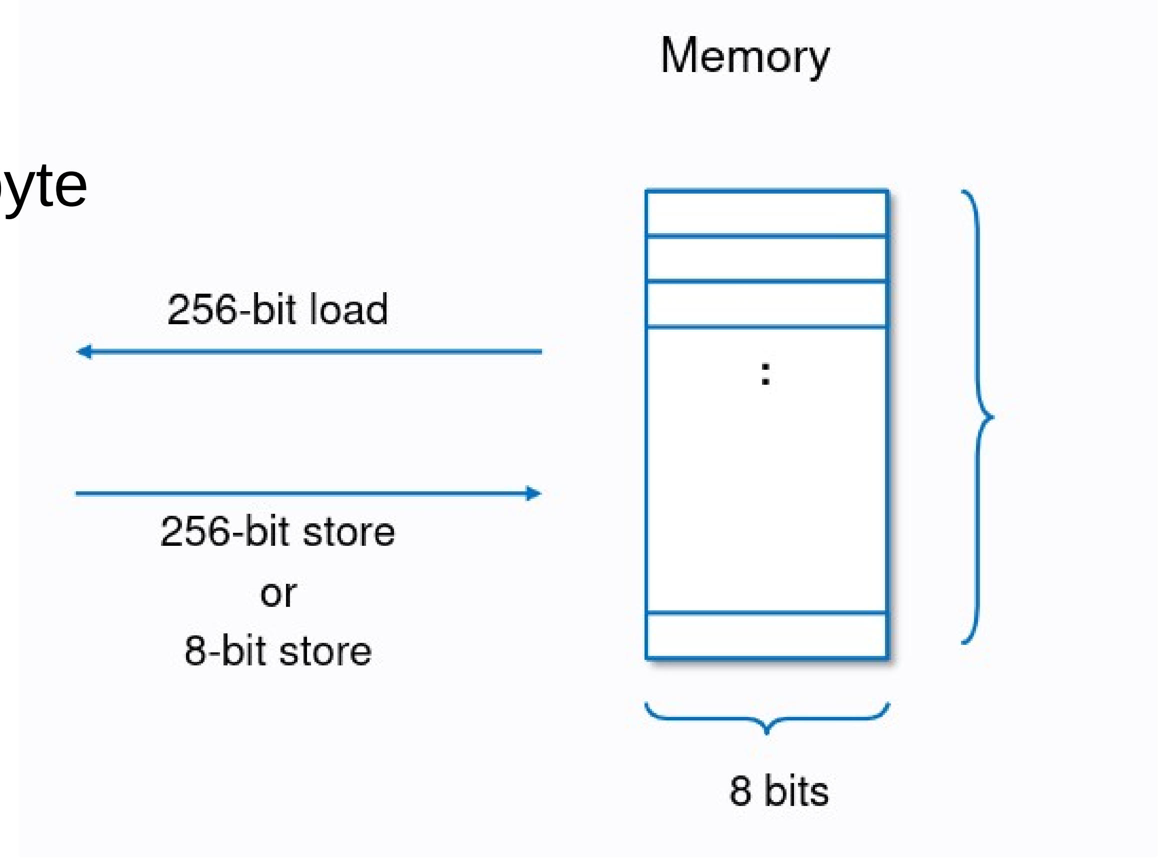
# Stack

- LIFO data structure
- PUSH (3 gas)
- POP (2 gas)



# Memory

- Byte array
- MSTORE: 32 byte / 1 byte
- MLOAD: 32 byte



# Memory opcodes

- Gas cost: 3 (static) + dynamic (memory expansion)

| | | | | |
|---|---|---|---|---|
| 0x51 | MLOAD | 1 | 1 | Load word from memory.<br>$\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \boldsymbol{\mu}_{\mathbf{m}}[\boldsymbol{\mu}_{\mathbf{s}}[0] \dots (\boldsymbol{\mu}_{\mathbf{s}}[0] + 31)]$<br>$\boldsymbol{\mu}'_{\mathrm{i}} \equiv \max(\boldsymbol{\mu}_{\mathrm{i}}, \lceil (\boldsymbol{\mu}_{\mathbf{s}}[0] + 32) \div 32 \rceil)$<br>The addition in the calculation of $\boldsymbol{\mu}'_{\mathrm{i}}$ is not subject to the $2^{256}$ modulo. |
| 0x52 | MSTORE | 2 | 0 | Save word to memory.<br>$\boldsymbol{\mu}'_{\mathbf{m}}[\boldsymbol{\mu}_{\mathbf{s}}[0] \dots (\boldsymbol{\mu}_{\mathbf{s}}[0] + 31)] \equiv \boldsymbol{\mu}_{\mathbf{s}}[1]$<br>$\boldsymbol{\mu}'_{\mathrm{i}} \equiv \max(\boldsymbol{\mu}_{\mathrm{i}}, \lceil (\boldsymbol{\mu}_{\mathbf{s}}[0] + 32) \div 32 \rceil)$<br>The addition in the calculation of $\boldsymbol{\mu}'_{\mathrm{i}}$ is not subject to the $2^{256}$ modulo. |
| 0x53 | MSTORE8 | 2 | 0 | Save byte to memory.<br>$\boldsymbol{\mu}'_{\mathbf{m}}[\boldsymbol{\mu}_{\mathbf{s}}[0]] \equiv (\boldsymbol{\mu}_{\mathbf{s}}[1] \bmod 256)$<br>$\boldsymbol{\mu}'_{\mathrm{i}} \equiv \max(\boldsymbol{\mu}_{\mathrm{i}}, \lceil (\boldsymbol{\mu}_{\mathbf{s}}[0] + 1) \div 32 \rceil)$<br>The addition in the calculation of $\boldsymbol{\mu}'_{\mathrm{i}}$ is not subject to the $2^{256}$ modulo. |

# Memory expansion

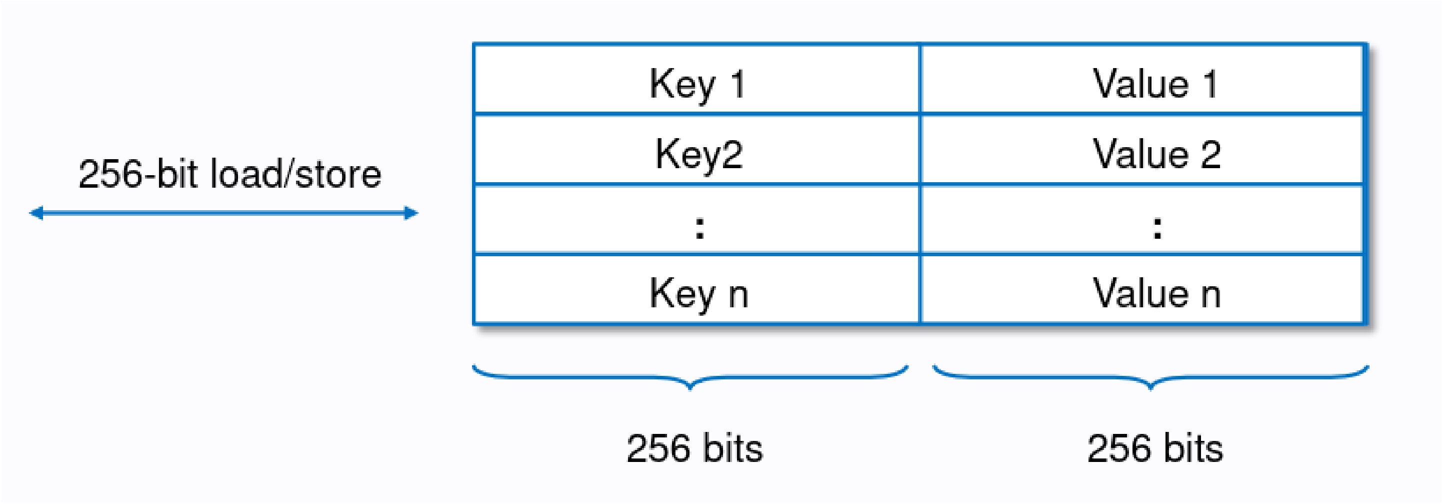
- When an offset is first accessed the memory is expanded in 32 byte chunks
- The cost of memory expansion grows quadratically:

$$C_{\mathrm{mem}}(a) \equiv G_{\mathrm{memory}} \cdot a + \left\lfloor \frac{a^2}{512} \right\rfloor$$

(`a` is the size of the memory, G_memory is 3)

# Storage

- 256 bit key-value store, all values initialized as zero
- Multiple variables can be stored in a single slot (storage packing)



# Storage: Gas

- Static cost of 100 gas for read (SLOAD) and write (SSTORE)
- Dynamic cost related to the state of the storage slot

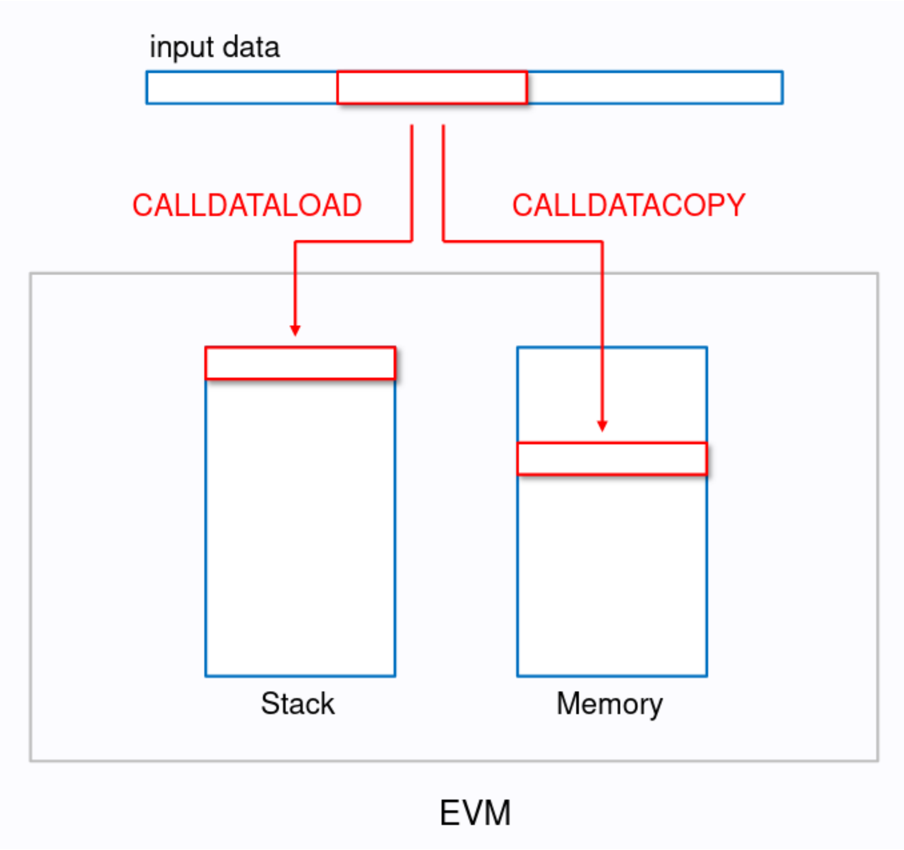
| | | |
|---|---|---|
| $G_{\text{warmaccess}}$ | 100 | Cost of a warm account or storage access. |
| $G_{\text{accesslistaddress}}$ | 2400 | Cost of warming up an account with the access list. |
| $G_{\text{accessliststorage}}$ | 1900 | Cost of warming up a storage with the access list. |
| $G_{\text{coldaccountaccess}}$ | 2600 | Cost of a cold account access. |
| $G_{\text{coldsload}}$ | 2100 | Cost of a cold storage access. |
| $G_{\text{sset}}$ | 20000 | Paid for an SSTORE operation when the storage value is set to non-zero from zero. |
| $G_{\text{sreset}}$ | 2900 | Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero. |
| $R_{\text{sclear}}$ | 15000 | Refund given (added into refund counter) when the storage value is set to zero from non-zero. |

# Storage: Gas

- Storage writes
  - zero to nonzero: 20k gas + 2.1k gas if first time access
  - Nonzero to nonzero: 5k gas
  - Nonzero to zero: refund up to 4.8k gas per storage variable
  - Zero to zero: 2.2k gas
- Storage reads
  - Cold: 2.1k gas
  - Warm: 100 gas

# Calldata

- Similar to memory, but read only
- Can only be used in external function calls



# Calldata: Gas

- Gas cost: 3 for calldataload/copy, 2 for calldatasize
- A byte of calldata costs 4 gas (if its zero) or 16 gas (nonzero)

| | | | | |
|---|---|---|---|---|
| 0x35 | CALLDATALOAD | 1 | 1 | Get input data of current environment. $\boldsymbol{\mu}'_s[0] \equiv I_{\mathbf{d}}[\boldsymbol{\mu}_s[0] \dots (\boldsymbol{\mu}_s[0] + 31)]$ with $I_{\mathbf{d}}[x] = 0$ if $x \geqslant \lVert I_{\mathbf{d}} \rVert$ This pertains to the input data passed with the message call instruction or transaction. |
| 0x36 | CALLDATASIZE | 0 | 1 | Get size of input data in current environment. $\boldsymbol{\mu}'_s[0] \equiv \lVert I_{\mathbf{d}} \rVert$ This pertains to the input data passed with the message call instruction or transaction. |
| 0x37 | CALLDATACOPY | 3 | 0 | Copy input data in current environment to memory. $\forall i \in \{0 \dots \boldsymbol{\mu}_s[2] - 1\} : \boldsymbol{\mu}'_m[\boldsymbol{\mu}_s[0] + i] \equiv \begin{cases} I_{\mathbf{d}}[\boldsymbol{\mu}_s[1] + i] & \text{if} \quad \boldsymbol{\mu}_s[1] + i < \lVert I_{\mathbf{d}} \rVert \\ 0 & \text{otherwise} \end{cases}$ The additions in $\boldsymbol{\mu}_s[1] + i$ are not subject to the $2^{256}$ modulo. $\boldsymbol{\mu}'_i \equiv M(\boldsymbol{\mu}_i, \boldsymbol{\mu}_s[0], \boldsymbol{\mu}_s[2])$ This pertains to the input data passed with the message call instruction or transaction. |

# References

- Ethereum EVM illustrated
- evm.codes
- Ethereum Yellowpaper
- Solidity internals